Privacy-Preserving Computation
(Position Paper)

Florian Kerschbaum

SAP Research
Karlsruhe, Germany
florian.kerschbaum@sap.com

Abstract. Private data is commonly revealed to the party performing the computation on it. This poses a problem, particularly when outsourcing storage and computation, e.g., to the cloud. In this paper we present a review of security mechanisms and a research agenda for privacy-preserving computation. We begin by reviewing current application scenarios where computation faces privacy requirements. We then review existing cryptographic techniques for privacy-preserving computation. And last, we outline research problems that need to be solved for implementing privacy-preserving computations. Once addressed, privacy-preserving computations can quickly become a reality enhancing the privacy protection of citizens.

1 Introduction

There are very strong privacy-enhancing techniques for communication and authentication available to citizens, but when it comes to actually processing the data few choices are available. Projects such as ANON and JAP [10] or TOR [20] enable anonymous communication. Tools such as IDEMIX [15] even enable anonymity-preserving authentication over these communication channels. Nevertheless, the data sent over anonymous (and authenticated) channels is rarely protected from the recipient.

This problem becomes prevalent when outsourcing storage and computation, e.g., to the cloud. Service providers such as Facebook or Google have made a business out of exploiting this data for advertising purposes. Big data is collected about habits and preferences of customers.

This problem is not due to a lack of available security mechanisms. Cryptography provides tools for privacy-preserving computations, such as secure multi-party computation, homomorphic encryption, order-preserving encryption or zero-knowledge proofs. In contrast to anonymous communication and authentication these mechanisms often lack a user-friendly implementation. This paper proposes a research agenda for implementing privacy-preserving computation bringing it closer to users' (and service providers') adoption. This paper does not argue that no further research in cryptography is necessary – quite the contrary, but existing mechanism should already be made available to system implementers. Furthermore, this paper focuses on the computational aspect,

B. Preneel and D. Ikonomou (Eds.): APF 2012, LNCS 8319, pp. 41–54, 2014.

i.e., securing data while computing, and not data privacy aspects dealt with k-anonymity [52] or differential privacy [21].

In the next section we present two exemplary scenarios where computation on private data is strictly necessary, but not desired or even illegal. In criminal investigations police institutions often need to exchange data about subjects, places or objects, but are (rightfully) restricted to necessary and proportionate cases. This may hinder investigation success. Privacy-preserving computation enables performing this data exchange, such that only data which is clearly linked to a related case is revealed, but other data is kept private. Innocent suspects are protected while criminal investigations are fostered.

In the future smart electricity grid household collect fine-grained consumption data and transmit it to the grid and utility providers. This data enables inferences about the household inhabitants and their preferences and represents a significant privacy invasion. In the Netherlands smart meter roll-outs have been stopped for this reason. Privacy-preserving computation enables smart meter data processing, such as billing, without revealing one's consumption data to the providers. It thereby reconciles the need for a smart grid with the privacy concerns of the citizens.

In Section 3 we review and compare a number of mechanisms for privacy-preserving computation. Secure multi-party computation [54] has been invented thirty years ago, but implementations are only beginning to emerge. Recently fully homomorphic encryption [22] has arisen as an alternative, but theoretical solutions are still to inefficient to be put to practice. Outsourced storage [11,45] can be efficiently processed if it is encrypted using searchable [14] or order-preserving encryption [3,13]. Data collected by cyber-physical sensors can be securely (and privately) processed using zero-knowledge proofs [26]. We will only gives a coarse characterization of techniques in order to judge their applicability in different scenarios.

Finally, in Section 4 we outline an agenda of tools that will help implement privacy-preserving computations and thereby increase user adoption. Currently, designing and implementing privacy-preserving computations is difficult. The programmer has to have problem-specific know-how for its application, programming experience to select the best implementation method and security awareness in order to ensure privacy. In a lot of cases the programmer has to make a choice between secure or privacy guarantees and efficiency. If he chooses to be efficient security needs to verified manually. Furthermore, every new computation (even if done at run-time) requires the same level of effort. We present a proposal for tools that should help remedy these problems.

2 Scenarios

We consider two exemplary scenarios where privacy needs to be protected during computation, i.e. they require privacy-preserving computation. These are criminal investigations and smart meter billing. Although both scenarios deals with computation on private data their most prominent solutions use different

techniques. Whereas criminal investigations are implemented using secure computation, smart meter billing is implemented using zero-knowledge proofs. This also highlights the applicability of different scenarios to different technologies. We will try to devise the characteristics of the scenarios lending them to specific technologies.

2.1 Criminal Investigation

In federated states or organization of states, such as the European Union or the United States, a common approach to organized crime is necessary. For this purpose, federal law enforcement agencies, such as Europol or the FBI, have been established. Nevertheless, data privacy laws (rightfully) restrict supplying institutions from sharing their data, unless there is a hard corroborating evidence on a case and subject under investigation.

A common tool for the criminal investigator is data mining using social network analysis of the data stored in their warehouses. It graphically depicts the suspects and their connections to other people or artifacts, such as telephone numbers or bank accounts, and allows the computation of certain metrics. Not all the facts composing the entire picture of a case may be known to one investigator. In particular, in pan-European organized crime, local police forces may only be aware of a partial view of the picture.

This necessitates data exchange between the institutions, but European data privacy laws restricts data exchange to necessary and proportionate cases. Therefore we propose a solution where the local investigator, or an investigator at the superordinate institution has access to all information, but without revealing sensitive or private details. This allows the investigator to still use SNA and profit from its achievements without breaking individual privacy rights or guidelines of other institutions.

Therefore privacy-preserving SNA – a special form of privacy-preserving data mining – has been suggested in the literature [39]. Each party inputs their view of the social network to a secure computation and the result is the anonymized combined view. No additional information, e.g., about unrelated suspects, is revealed, i.e., their privacy is being preserved.

Note that in order to compare for identical entries in the social network – a computation on joint data is necessary. No party can by itself decide whether the data of the other party matches its own. It is therefore important to note that in this scenario both parties provide (private) input.

For the practical adoption of privacy-preserving computation in criminal investigations several legal, political and social considerations have to be taken into account. We argue that the ready-to-use availability of the technology will spur the public discussion on these topics.

2.2 Smart Meter Data

Smart metering refers to the collection of consumption profiles at customer's households with the help of so called smart meters (SM). Smart meters measure

electricity consumption in households and communicate their readings at regular intervals to the back-end system. Alternatively, the back-end system can also query the smart meter for its data (pull). A Trusted Platform Module in the smart meter holds key material and creates signatures over the data to ensure authenticity and integrity until it arrives at the back-end system. There the consumption profile and the tariff data from the respective customer's contract are used to calculate the price the customer has to pay for the time period covered by the profile.

Smart metering has encountered massive privacy concerns from media [35], data privacy experts [16] and consumers [30]. The fact that whole consumption profiles of households are transmitted to and stored by suppliers is troubling w.r.t. customer privacy. Data confidentiality can be easily protected in transit between smart meter and back-end system. However, their storage at the suppliers' IT-systems still endangers customer privacy. Depending on resolution and the availability of different services' profiles (e.g. water, heat, electricity) one can read the profile and "see" more or less clearly what happens in the household: For instance, when family members wake up (light switched on), whether they shower in the morning (water, heat, and electricity for water heater), whether they drink hot beverages with their breakfast and when or if they leave for work or school. Furthermore, the frequency of washing and drying clothes, cooking or the amount of time the TV is turned on can be inferred. For further research on what electricity consumption profiles tell about the inhabitants see [7,29,40,51].

These inferences make consumption profiles very privacy-sensitive data and these profiles might even have value in the advertising market, for instance. On one hand, disgruntled employees or external attackers might attempt to steal it for profit or out of malice. On the other hand, the supplier could seek subsidiary revenues by selling this data himself. Depending on the local jurisdiction, this might even be legal.

The important point is, that currently there are no reliable, technical measures in place to prevent abuse of consumption profiles. Merely organizational measures, policies or laws sanction the abuse of privacy related data but require a trace or proof of abuse and do not prevent it in the first place.

First [36] and later independently [46] introduce a privacy component into the standardized smart meter / meter data management (MDM) reporting communication link. This component hides the actual consumption profile from the MDM and therefore also from the supplier. This only requires small changes compared to current smart meter reporting. The privacy component intercepts smart meter readings, then uses tariff information provided externally (over the Internet or by the MDM) to calculate the billing amount and sends only the resulting billing amount to the MDM.

The technical solution in this scenario is a Zero-Knowledge Proof (ZKP). The user proves to the service provider that it truthfully computed the bill. This works for two reasons: First, the user only computes on his private data, i.e., there is no second party (private) input. Second, the integrity and authenticity

of the data is ensured by trusted components. The smart meters can digitally sign their measurements and are implemented with trusted hardware.

We envision this scenario setup to be quite common in cyber-physical systems. It has already been described by Danezis and Livshits in a related position paper for cloud computing [19]. Sensors collect data about people. Obviously, this data may be privacy-sensitive and protected by privacy legislation. These sensors can ensure the integrity of the collected data, but should not reveal it. Using a ZKP the user can perform the computation itself in a privacy-preserving fashion and prove correct computation. Other examples of this scenario setup include, e.g., road toll pricing [6] or e-ticketing in public transportation [32].

For the practical adoption of this technology a user device (plug-in component) needs to be part of the protocol [36]. While the business implications of this design are not field-tested, we argue that increased privacy of the consumer might make an interesting business case.

3 Security Mechanisms

In this section we briefly and mostly non-technically review the mechanisms of secure computation, homomorphic encryption, order-preserving encryption and zero-knowledge proofs. We address the properties of each in terms of security, performance and functionality. Balancing these three objectives is the challenge of privacy-preserving computation. Particularly, we highlight their suitability for certain application scenarios.

3.1 Secure Multi-party Computation

Yao introduced secure two-party computation in [54]. Secure (two-party) computation allows two parties to compute a function f over their joint, private inputs x and y, respectively. No party can infer anything about the other party's input (e.g. y) except what can be inferred from one's own input (e.g. x) and output (e.g. $f(x, y)$).

Yao's initial protocol uses a technique called garbled circuits. Alice prepares a circuit for the function to be computed and encrypts (and garbles) this circuit. The encrypted circuit is transferred to Bob who obtains keys for his input using oblivious transfer. Bob then decrypts (part of) the circuit obtaining the function result. For a detailed, technical description of circuit garbling and its implementation see [42].

Now, there are many protocols for secure computation. They can be classified into generic and special protocols. Generic protocols can implement any functionality whereas special protocols implement one specific function. Generic protocols contain a translation step for the function – similar to Yao's garbled circuit construction. Their security proof, however, is independent of the function. Special protocols are usually more efficient, since they use problem insight to optimize the protocol. They need to proven secure manually for each protocol instance.

For a very long time generic protocols exist for multi-party computations in the computational [25] and information-theoretic setting [9]. Also they exist for many different security models. Most notable are the semi-honest and malicious security models [24]. In the semi-honest model the parties are assumed to follow the protocol. In the malicious model they may deviate arbitrarily. The policy implications of these models need to be discussed in the context of the concrete use case and its actors, e.g., criminal investigations. A detailed investigations is subject to further research and out of scope of this position paper.

Security of secure computations is often defined by comparison to an ideal model. In the ideal model there is a trusted third party. All parties send their inputs to the trusted third party which computes the function and returns the result to the parties. The function implemented by the third party is also ideal functionality. Each attack feasible in the real protocol execution must also be feasible in this ideal model.

It is important to note that neither model prevents inferences about the input from the result. This may be particularly sensitive if one party may influence the function to be computed. For example, assume that one may party may privately issue a query about the other's party private database. Then it is hard to preserve the privacy of the database without additional measures such as differential privacy [21]. Furthermore, no security model prevents the substitution of inputs. Therefore an interest in the correct computation of the result needs to be assumed. This is subject to economic security models, such as non-cooperative computation [50]. Even when implementing a non-cooperative computation secure computation may be difficult to implement [2,27,28]. Nevertheless using specialized protocols it can even be implemented for mixed security models [41]. Implementing a proxy, e.g., using cloud computing, may help solve the problem [37]. In theory, it is possible to implement any computation using rational players [34], but it requires physical assumptions.

There exist a number of domain-specific programming languages for implementing secure computations [8,12,18,31,33,42,48]. They can be classified into those tied to a generic protocol [8,12,31,42] or those based on generic programming languages [18,33,48]. The second kind can implement a wider variety of protocols, but also enables implementing insecure protocols. The ones tied to a specific protocol may be proven secure independent of the functionality. Such a proof extends to all protocols implemented in this language, but the language prevents implementing many special, possibly more efficient protocols.

We classify them into systems specifying the ideal functionality and systems specifying the protocol description. Just FairPlay [42] and FairPlayMP [8] are instances of systems which only describe the ideal functionality of a secure computation, i.e. *what* is to be implemented by the protocol. All of the other languages, compilers or frameworks are instances of systems where the programmer can – at least partially – specify *how* the protocol is implemented. This approach leads to significantly more efficient protocols, but puts an additional burden on the programmer.

3.2 Homomorphic Encryption

Homomorphic encryption supports a homomorphism of (at least) one arithmetic operation on the ciphertexts to an arithmetic operation on the plaintexts. Additively homomorphic operation supports addition as the homomorphic operation on the plaintexts. Let $E(x, r)$ denote the encryption of plaintext x with randomization parameter r. Then the following addition properties hold

$$D(E(x)E(y)) = x + y$$

The most popular additively homomorphic encryption system is Paillier's [44]. It is public key and satisfies modern security definitions (semantic security). Its performance is comparable to other public key encryption systems.

Gentry recently developed a fully homomorphic encryption scheme [22]. It supports both, addition and multiplication, and therefore allows the computation of arbitrary functions on the ciphertext. Nevertheless, its performance is severely restricted in practice [23] and more efficient schemes are still subject to research.

As middle-ground there are somewhat homomorphic encryption schemes. They support a limited number of multiplications and thereby enable to compute a wider class of functions than purely additively homomorphic encryption. Their performance is comparable to additively homomorphic encryption [43].

A fundamental advantage of homomorphic encryption compared to secure computation is its non-interactivity. The client submits input and a server can perform the computation without learning anything but ciphertexts. The result is a ciphertext as well. Therefore computations on homomorphic encryption can be performed off-line and do not depend on the network performance. The client only has to communicate data linear in its input length whereas in secure computation it is linear in the function's complexity.

Nevertheless, besides its performance homomorphic encryption also has a number of limitations. First, the result and each intermediate result is encrypted. Therefore the server cannot make any decision based on its value. Similar to secure computation the server therefore needs to compute over all choices, i.e., in every conditional branch both branches need to be evaluated. This increases the complexity of each function to its worst case complexity and further increases the performance penalty.

More severely, computations on homomorphic encryption must be performed under the same key. It has been proven in [53] that no fully homomorphic encryption scheme with multiple keys can exist. This implies a collusion attack for collaborative computations. When two parties submit input, one party only holds the public key. The input data of this party may be decrypted by the private key holder, e.g., by a collusion with the server. Homomorphic encryption therefore seems less suitable for collaborative computations.

Additively homomorphic encryption can also be used to implement secure computation [17]. This makes the computation interactive again, but prevents the collusion attack. Usually secure computation based on homomorphic encryption is less efficient than other generic protocols [47].

3.3 Order-Preserving Encryption

Order-preserving encryption [3,13] ensures that the order (greater-than relation) of the ciphertexts is the same as the order of the corresponding plaintexts. This allows a server to efficiently search on the ciphertexts using binary search or perform range queries. In turn this capability enables performing most database queries on encrypted data [11,45].

Efficient encrypted database – where the key is stored outside of the database – have many applications in privacy-preserving computing. Cloud computing using a database-as-a-server model can be secured against the service provider such that insider or targeted attacks are significantly complicated. As such real-world applications of prototypical system have been already reported [11].

The efficiency gain of order-preserving encryption mainly stems from its main difference to the homomorphic encryption. The result of the computation on the ciphertexts is publicly available to the server performing the computation. This enables implementing significantly more efficient algorithms.

Another instance of such an encryption scheme is searchable encryption [14]. Searchable encryption allows to compare for equality of plaintexts using a token issued by the private key holder. Differently from order-preserving encryption it can be proven secure in more standard security models. The best security proof for order-preserving encryption is that it is as secure as possible under the order-preserving constraints [13]. How secure this level of security is still subject to research. Searchable encryption has been proven secure against chosen plaintext attacks.

Searchable encryption requires a linear scan over the data, i.e., an index is useless. It therefore has not found the same acceptance in the database community as order-preserving encryption, although it also allows range queries [49].

Clearly, since the result of the comparison is revealed, this type of encryption is limited to specific functions. First, the size of the ciphertexts needs to remain manageable. Second, certain functions may allow breaking the encryption scheme. For example, a public-key order preserving encryption scheme would allow binary search for arbitrary plaintexts. It cannot be secure. Therefore all order-preserving encryption schemes are symmetric.

Due to the limitation of the functionality, order-preserving encryption has limited applications. A secure, encrypted database is certainly a great achievement, but more complicated applications seem to be difficult to design. They may not be securely implementable and if they are, then their construction can be very complicated.

3.4 Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKP) have been introduced in cryptography a long time ago [26]. They allow the proof of knowledge of some data that satisfies a certain function. In the example of smart meter data, this function is the signatures by the smart meters and the billing amount. The household then proofs knowledge of consumption data that is signed and amount to the bill.

ZKPs in the first place ensure integrity, i.e., the function has been computed truthfully. As such their application to privacy-preserving computation is not obvious, but ZKPs allow the outsourcing of the computation while verifying the integrity of the computation. Therefore they allow the outsourcing the originator of the (private) data, although it might not be trusted. At least it can ensure the confidentiality (privacy) of the data. The ZKP ensures that it cannot cheat.

Consequently, ZKPs are applicable in settings where data is collected about a person, such as smart metering or road-toll pricing. ZKPs are similarly limited as homomorphic encryption when it comes to input of multiple parties. Then secure computation is the method of choice, but ZKPs can also be implemented non-interactively. Nevertheless due to the emergence of more and more cyber-physical systems where sensors collect data about person ZKPs will probably enjoy wider adoption.

The initial ZKPs were generic allowing to proof for any function in NP. Then some special ZKPs were designed, e.g., for discrete logarithms or range proofs. Recently, compilers translating function descriptions into ZKPs have been presented [4,5].

ZKPs are also the basis for privacy-preserving authentication. A user can proof the possession of an attribute - such as age or driver's license – without revealing any private information. Furthermore, all transaction may remain unlinkable. Therefore the future adoption of ZKPs to privacy-preserving computation seems even more likely. We recommend to also consider the position paper [19] by Danezis and Livshits which already outlines similar ideas on ZKPs.

4 Research Agenda

In order to foster the adoption of privacy-preserving computation for many more applications there need to be tools. These tools need to ease the design and development of privacy-preserving computing applications. Currently, there is a lot of manual effort in developing a privacy-preserving application. This hinders adoption in practice due to a lack of skills and available resources. Development tools can significantly lower the barrier of adoption. Other characteristics such as security (privacy) and even performance can be made to fit, if the system is designed cleverly.

4.1 Compiler

As mentioned before there already exist some compilers for secure computation and zero-knowledge proofs. These compilers unfortunately currently still fall short of the requirements of the developers. We will highlight some design principles, challenges and ideas for achieving the full potential. These design principles should be seen as visionary, ambitious objectives and even partial, but rigorous achievement can be viewed as scientific success.

I. *Only the ideal functionality should be specified.*

The principle of FairPlay shows the right direction. It is already difficult enough for the programmer to acquire problem domain-specific knowledge in order to design a successful application. If the programmer also has to care about security (and complex performance aspects), he will be overburdened. The compiler has to take care of it.

A necessary additional specification is the data origin, i.e., which party provides which input. There may be a need for some security policies, but largely extending current access control. Privacy-preserving computation allows to enforce policies, such as security against a service provider. There is no longer a grant or deny decision.

In the system architecture there may also be a need to specify trust relationships. Nevertheless, there should not be a restriction to or reliance on specific trust assumptions. Instead, privacy-preserving can reduce or even remove most of the commonly necessary trust assumptions.

II. *Security should be guaranteed.*

Except for the specification of policies and trust assumption security must be guaranteed. Every successfully compiled program must be secure. The definition of security must depend on the policy and the security model (which may also be specified as part of the policy). Ideally, the compiler generates a proof or is certified that the compiled program is secure. We have seen for secure computations that some domain-specific languages violate this principle for security reasons.

There are several security models available in the cryptographic literature, but sometimes they may fall short in capturing the system's dependencies. For example, it is easy to design a protocol that is secure in the semi-honest protocol, if all inputs can be revealed by the result of the function. Therefore additional tools analyzing the functionality for admissible information flow in the entire system are necessary. Only, if the programmer is not capable of "shooting himself in the foot", a system can be considered simple to implement and secure. Therefore, there is a clear need for security models and tools augmenting the available cryptographic security models.

Language annotations, such as type systems, may significantly simplify this problem. E.g. type systems for information flow [1] allow only specifying programs that do not contain non-admissible information flows. Type systems can encode any type of security proof, but the research challenge remains to analyze the admissible information flows.

III. *Performance optimization should be automatic.*

Out of the set of admissible protocols according to the security model the compiler should select the best performing one. This selection should be automatic, i.e., without the need of specification by the programmer. There are several challenges and approaches to this problem.

First, the compiler may use many of the algorithms available to the compiler community. Algorithms like data flow or program analysis need to be augmented for additional criterion – security. We see a number of results in this area, e.g., in secure computation optimization [38].

Second, there is a need for a clear performance model. The typical complexity measure of algorithms do not fit any more, since there is this additional criterion of security. Let n be the input length, then we also have the security parameter k. Now, what is faster $O(n^2k)$ or $O(nk^2)$? Performance models can make decision for this and are already often used in other compilers, e.g., SQL query optimization.

Given a comprehensive performance model, the programmer should no longer need to specify the type of mechanism to be used. Instead, based on analysis of the data model and the function the compiler should select the optimally suitable one, e.g., secure computation or homomorphic encryption. Admittingly, some methods, such as order-preserving encryption, can be hard to fit into this model, particularly due to their unclear security model.

Based on the problem of complicated complexities the programmer may actually implement a sub-optimal algorithms, because he is not aware of all information for the decision. Therefore the compiler should be equipped with capabilities to rewrite programs, such that they perform better. Some rewriting techniques, e.g., common sub-expression elimination, have been developed in compiler design. Again, these need to be augmented by a security criterion. Furthermore, since there is no programming language established yet, we can also adapt the design of the language.

5 Conclusions

In this paper we investigated the status and future challenges of privacy-preserving computation. We have the security mechanisms available, but face the challenge of implementing them. Performance has already been proven in several applications not to be the prohibiting factor anymore and available computational resources continue to increase.

The problem will shift to the development of the applications. Current tools do not scale to the expected increase in privacy-preserving computation. We outlined some exemplary application which can serve as blue prints for others waiting to be implemented.

We then outlined the research challenges for a compiler for privacy-preserving computation. Based on three principles relating to the three objectives of privacy-preserving computation we described some research challenges and approaches. The purposes of this agenda is entice discussion and interest among interdisciplinary stakeholders in order to foster the adoption of privacy-preserving computation. Seeing what will be possible may lift some of the prejudices privacy-preserving computation currently faces. Ultimately only the uptake of technology will lead to a better protection of the citizen's privacy.

References

1. Abadi, M., Morrisett, G., Sabelfeld, A.: Language-based security. Journal of Functional Programming 15(2), 129 (2005)
2. Abraham, I., Dolev, D., Gonen, R., Halpern, J.: Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In: Proceedings of the 25th ACM Symposium on Principles of Distributed Computing, PODC 2006 (2006)
3. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of the ACM International Conference on Management of Data, SIGMOD 2004 (2004)
4. Almeida, J.B., Bangerter, E., Barbosa, M., Krenn, S., Sadeghi, A.-R., Schneider, T.: A certifying compiler for zero-knowledge proofs of knowledge based on σ-protocols. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 151–167. Springer, Heidelberg (2010)
5. Backes, M., Maffei, M., Pecina, K.: Automated synthesis of privacy-preserving distributed applications. In: Proceedings of 19th Network and Distributed System Security Symposium, NDSS 2012 (2012)
6. Balasch, J., Rial, A., Troncoso, C., Preneel, B., Verbauwhede, I., Geuens, C.: Pretp: privacy-preserving electronic toll pricing. In: Proceedings of the 19th USENIX Conference on Security, USENIX Security 2010 (2010)
7. Bauer, G., Stockinger, K., Lukowicz, P.: Recognizing the use-mode of kitchen appliances from their current consumption. In: Barnaghi, P., Moessner, K., Presser, M., Meissner, S. (eds.) EuroSSC 2009. LNCS, vol. 5741, pp. 163–176. Springer, Heidelberg (2009)
8. Ben-David, A., Nisan, N., Pinkas, B.: Fairplaymp: a system for secure multi-party computation. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS 2008 (2008)
9. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the 20th ACM Symposium on Theory of computing, STOC 1988 (1988)
10. Berthold, O., Federrath, H., Köhntopp, M.: Project "anonymity and unobservability in the internet". In: Proceedings of the 10th Conference on Computers, Freedom and Privacy: Challenging the Assumptions, CFP 2000 (2000)
11. Binnig, C., Hildenbrand, S., Färber, F.: Dictionary-based order-preserving string compression for main memory column stores. In: Proceedings of the ACM International Conference on Management of Data, SIGMOD 2009 (2009)
12. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008)
13. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009)
14. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
15. Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002 (2002)
16. Cavoukian, A., Polonetskyand, J., Wolf, C.: Smart privacy for the smart grid: embedding privacy into the design of electricity conservation. Identity in the Information Society 3(2), 275–294 (2010)

17. Cramer, R., Damgård, I.B., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)
18. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (2009)
19. Danezis, G., Livshits, B.: Towards ensuring client-side computational integrity (position paper). In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW 2011 (2011)
20. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: Proceedings of the 13th USENIX Conference on Security, USENIX Security 2004 (2004)
21. Dwork, C.: Differential privacy. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (2006)
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st ACM Symposium on Theory of Computing, STOC 2009 (2009)
23. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)
24. Goldreich, O.: The Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press (2004)
25. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the 19th ACM Symposium on Theory of Computing, STOC 1987 (1987)
26. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal of Computing 18(1), 186–208 (1989)
27. Gordon, S.D., Katz, J.: Rational secret sharing, revisited. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 229–241. Springer, Heidelberg (2006)
28. Halpern, J., Teague, V.: Rational secret sharing and multiparty computation: Extended abstract. In: Proceedings of the 36th ACM Symposium on Theory of Computing, STOC 2004 (2004)
29. Hart, G.W.: Nonintrusive appliance load monitoring. Proceedings of the IEEE 80(12), 1870–1891 (1992)
30. Heck, W.: Smart energy meter will not be compulsory. NRC Handelsblad (April 2009), http://www.nrc.nl/international/article2207260.ece/Smart_energy_meter_will_not_be_compulsory
31. Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: Tasty: tool for automating secure two-party computations. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010 (2010)
32. Heydt-Benjamin, T.S., Chae, H.-J., Defend, B., Fu, K.: Privacy for public transportation. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 1–19. Springer, Heidelberg (2006)
33. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proceedings of the 20th USENIX Conference on Security, USENIX Security 2011 (2011)
34. Izmalkov, S., Micali, S., Lepinski, M.: Rational secure computation and ideal mechanism design. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science, FOCS 2005 (2005)
35. Jamieson, A.: Smart meters could be 'spy in the home'. Telegraph (UK) (October 2009), http://www.telegraph.co.uk/finance/newsbysector/energy/6292809/Smart-meters-could-be-spy-in-the-home.html

36. Jawurek, M., Johns, M., Kerschbaum, F.: Plug-in privacy for smart metering billing. In: Fischer-Hübner, S., Hopper, N. (eds.) PETS 2011. LNCS, vol. 6794, pp. 192–210. Springer, Heidelberg (2011)
37. Kerschbaum, F.: Adapting privacy-preserving computation to the service provider model. In: Proceedings of the International Conference on Privacy, Security, Risk and Trust, PASSAT 2009 (2009)
38. Kerschbaum, F.: Automatically optimizing secure computation. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011 (2011)
39. Kerschbaum, F., Schaad, A.: Privacy-preserving social network analysis for criminal investigations. In: Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society, WPES 2008 (2008)
40. Lisovich, M.A., Mulligan, D.K., Wicker, S.B.: Inferring personal information from demand-response systems. IEEE Security and Privacy 8(1), 11–20 (2010)
41. Lysyanskaya, A., Triandopoulos, N.: Rationality and adversarial behavior in multi-party computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 180–197. Springer, Heidelberg (2006)
42. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay – a secure two-party computation system. In: Proceedings of the 13th USENIX Conference on Security, USENIX Security 2004 (2004)
43. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In. In: Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011 (2011)
44. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
45. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles, SOSP 2011 (2011)
46. Rial, A., Danezis, G.: Privacy-preserving smart metering. In: Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society, WPES 2011 (2011)
47. Schröpfer, A., Kerschbaum, F.: Forecasting run-times of secure two-party computation. In: Proceedings of the 8th International Conference on Quantitative Evaluation of Systems, QEST 2011 (2011)
48. Schröpfer, A., Kerschbaum, F., Müller, G.: L1 – an intermediate language for mixed-protocol secure computation. In: Proceedings of the 35th IEEE Computer Software and Applications Conference, COMPSAC 2011 (2011)
49. Shi, E., Bethencourt, J., Chan, T.-H.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: Proceedings of the IEEE Symposium on Security and Privacy, SP 2007 (2007)
50. Shoham, Y., Tennenholtz, M.: Non-cooperative computation: boolean functions with correctness and exclusivity. Theoretical Computer Science 343(1-2), 97–113 (2005)
51. Sultanem, F.: Using appliance signatures for monitoring residential loads at meter panel level. IEEE Transactions on Power Delivery 6(4), 1380–1385 (1991)
52. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5) (2002)
53. Van Dijk, M., Juels, A.: On the impossibility of cryptography alone for privacy-preserving cloud computing. In: Proceedings of the 5th USENIX Workshop on Hot Topics in Security, HotSec 2010 (2010)
54. Yao, A.C.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, FOCS 1982 (1982)